

# A General Logic Structure for Custom LSI

M. W. Sievers

Communications Systems Research Section

*A designer of custom large-scale integrated circuits (LSI) should be primarily concerned with deriving a working chip as rapidly as is practical. Obtaining maximal usage of area or pushing a technology to its limit is best left for large producers of LSI who are able to recover the additional required expenditures of time and processing facilities by volume sales. Custom design may be greatly facilitated by a general template into which most circuitry can be built. This template must be both simple to use and reasonably conservative of area to be practical.*

*This paper describes a general structure that is being considered as a template candidate. Examples of circuits built in NMOS technology are shown. It is hoped this structure is suitable for building special-purpose devices such as correlators and FFT's as well as general-purpose controllers.*

## I. Introduction

This paper describes an integrated circuit structure suitable for implementing general combinatoric logic functions in custom LSI. The general logic structure (GLS) is similar to a programmed logic array (PLA) with additional features that enhance its flexibility. GLS was motivated by the storage logic structure described by Patil (Ref. 1).

Now the bad news. It can be expected that the mapping of an arbitrary logic function into a general structure is less efficient in use of area and probably slower than an optimized random logic design. However, in the realm of custom LSI, these factors may not be significant, especially if they may be traded for implementation ease and reduced cost. It should be noted further that this structure may not be suitable for

all possible circuitry. Some elements, most notably memory, pad drivers, and line receivers with Schmitt trigger inputs are best built outside of the GLS. The good news is that there are very few classes of these special circuits.

General structures are also attractive in fault tolerant designs. This is because once the failure mechanisms have been established for the structure, this information may be used to predict the reliability of functions mapped into it and to suggest design rules to increase fault and test coverage.

## II. GLS Description

An approximation of the logical equivalent of a GLS is shown in the example of Fig. 1. Each of the  $m$  column lines

may be formed into up to two NOR gates from two physically disjoint partitions of the  $n$  implicant rows. A gate is created by connecting pull-down transistors at desired implicant rows to a column and connecting the column to a pull-up resistor. The output of the NOR gate may be taken anywhere on the column segment on which the gate is built. Pull-up resistors are provided at both ends of each column. This permits a column to be cut and formed into two independent NOR gates.

A NOR gate output may be connected to any unused implicant row that it crosses. The connection is made by forming a contact point at the desired row-column intersection. Formation of multiple contact points allows transmitting the NOR output to different locations within the structure. Distribution of signals is further facilitated by connecting implicant rows directly to columns. In this way, an implicant carried at the "top" of the GLS may be brought to an implicant row at the "bottom."

Implicant rows may be cut into several segments. Each segment can be used for transmitting a different implicant. In this way, for example, an implicant row carrying an input term may be cut after the last pull-down transistor that uses it. The segment after the cut can then be reused to transmit a NOR output.

The example of Fig. 1 illustrates some of the flexibility of the GLS. Contact points are indicated by a dot and cuts by an x. NOR gate 1 forms the result  $\overline{A+B+C}$ . The implicant row carrying input A is cut at Z and a contact point is created at Y. This connects gate 1 output to the implicant row freed by cut Z. Gate 1 output is now available as an input to NOR gate 3. Gate 3 output is connected to an unused implicant row which in turn is connected at T to a column. Contact W places the output of gate 3 on an implicant row, which makes it available to the input of gate 2.

Figure 2 continues the example of Fig. 1. Some of the abstraction used to depict the GLS in Fig. 1 has been elided here to more clearly represent its actual implementation. The implementation shown is based in NMOS technology.

Two types of rows are shown in Fig. 2. One row type transmits implicant terms and is drawn as a solid line. The other row type is connected to ground and is drawn as a dashed line. The reason for the two row types will become clearer in the discussion of Fig. 3. Columns are drawn as alternate dot-dashed lines.

Transistors are created at the intersection of a column and implicant row where the implicant is needed in the NOR function. Transistor channels provide a switchable path from

columns to the ground row. Pull-down transistors are enhancement mode devices. The channels of these transistors conduct when a positive gate-to-source voltage is present.

Pull-up resistors are implemented by depletion mode transistors that are shorted from gate to source. In this configuration, the transistor acts like a resistor with an impedance proportional to its length-to-width ratio.

When an implicant is high, it causes all transistors to which it is connected to conduct. This forces the corresponding columns to approximately the ground potential. When none of the implicants associated with a given column is high, the pull-up resistor brings the column to approximately the positive supply voltage; a NOR function.

The example of Fig. 1 is revisited one more time in Fig. 3. Figure 3 illustrates the stick diagram for an NMOS implementation of Fig. 1. In this figure, polysilicon is represented by a solid line, diffusion by a dashed line, metal by an alternating dot-dashed line, and implant by a thickened line.

When polysilicon crosses diffusion, a transistor is formed in which the polysilicon becomes the gate and diffusion the channel. This phenomenon is exploited in the GLS in the same manner as in an NMOS PLA. Polysilicon is used to transmit all implicant terms across the structure. Transistor channels, connecting metal columns to grounded diffusion rows are made to cross polysilicon rows of the implicants needed in the NOR to be formed. Formation of transistors in this manner necessitates an interleaving of polysilicon implicant rows and grounded diffusion rows.

The GLS illustrated in Figs. 1 thru 3 shows only two pull-up resistors per metal column. This restriction is unnecessary as shown in Fig. 4. The technique for providing additional pull-ups requires that the metal columns be periodically broken. The resulting gap is spanned by a diffusion cross-under. This diffusion is crossed by a metal carrying the positive supply voltage.

Three situations occur as represented by A, B, and C in Fig. 4. Situation A illustrates the method for providing pull-up resistors at column breaks. Polysilicon and implant are placed as shown to form depletion mode transistors. The poly gate is fed around to the transistor source and a contact point is made that connects the metal column, poly, and source. The drain side of the transistor channel is connected to the metal row carrying the positive supply voltage. Example B illustrates the formation of a communication path between two column pieces. The column pieces are connected by creating contact

points to the diffusion cross-under. Finally, example C shows that disjoint column pieces are made by simply omitting contact points from metal to the cross-under diffusion.

### III. Lower Bound on the Size of an NMOS GLS Binary Tree

An estimate of the lower bound on the size of a binary tree implemented in an NMOS GLS may be found with the aid of the binary NOR tree in Fig. 5. For this discussion, assume that the tree is complete; i.e., there exists an integer  $m$  such that the number of inputs  $n$ , is equal to  $2^m$ . This is the simplest tree that can be formed in the GLS because the GLS naturally forms NOR gates. Clearly, although any tree can be mapped into a binary NOR tree, trees with irregular structure require more space.

The complete  $n$  input binary NOR tree in Fig. 5 is formed from a total of  $n - 1$  NOR gates. Owing to the natural signal flow in the tree, it is best to build the tree exactly as shown in Fig. 5. That is, as a series of columns, each column containing a single level of gates. Since the depth of the tree is  $\log_2 n$ , that many columns are needed. Since the output of any gate may be placed on one of the implicant rows which drives the gate, no proliferation of implicant rows occurs. Therefore, at most,  $n$  implicant rows are needed. The tree requires  $n - 1$  gates so the total area occupied by the tree will be  $(n - 1)(G + P)$ , where  $G$  is the area of a gate and  $P$  is the area of the pull-up resistor.

Estimating the size of other trees formed in a GLS may be done by first determining the equivalent binary NOR tree. The total number of metal columns needed will be no less than the base 2 logarithm of the number of inputs; the number of rows will be no less than number of inputs, and the area will be at least one less than the number of inputs times the area of a gate and its pull-up.

### IV. Registers

In order for the GLS to be completely general it is necessary to demonstrate that it is feasible and practical to implement registers in it. One possible register is shown in Figs. 6a and 6b. Figure 6a shows the logical equivalent of a D flip-flop built with NOR gates. The storage element in this device is constructed from cross-coupled NOR gates. When both inputs of the flip-flop are low, the output remains unchanged. A high level signal to either input will force the output of the corresponding NOR gate low. Both outputs will go low if both inputs are high, but the state of the flip-flop is indeterminate if both inputs go low again simultaneously.

An input gating arrangement enables the data input to the flip-flop. When the clock is high, the flip-flop output will remain at its current state. When the clock goes low, the output of the flip-flop follows the input. Data is latched into the flip-flop when the clock returns to its high state.

An NMOS stock diagram of the register in Fig. 6a is shown in Fig. 6b. The line convention is the same as in Fig. 3.

This register fulfills the clocking requirements for testability established in Ref. 2. Complete compatibility with those requirements necessitates that all registers be shift registers. Placing a multiplexer (see Fig. 7 for an example of a multiplexer implemented in NOR gates) in front of the flip-flop is one possible method for building a shift register. The multiplexer selects data either from a previous register, the next register, a direct input, or the current data. In this configuration, the register may be continuously clocked. The multiplexer determines whether a left or right shift, load, or no operation is to occur on the next clock transition. A stick diagram of a two cell shift register is shown in Fig. 8.

### V. A Few Remarks on Optimality

A completely optimum design belongs in the same class of mythical creatures as unicorns, tax reform, and honest used car salesmen. It is possible to speculate on its existence, but it is impossible to achieve or recognize in any nontrivial situation. Some comments can be made, however, regarding "best" use of area when a GLS is compared with a PLA or gate array, and in "best" selection of architecture when a choice exists.

In a comparison of a PLA and GLS implementation of a given logical expression, it should be expected that the GLS will be more dense on the average. In a PLA, the concept of optimum design is restricted to finding the minimum set of implicants required to compute a given function. The GLS permits an added dimension by allowing a number of cut-and-paste operations that can be used to reclaim some of the area that would otherwise be lost in PLA designs. There will no doubt be functions that map as inefficiently into a GLS as a PLA. It is hoped that these are rarely encountered.

The density of functions mapped into gate arrays will also generally be less than that achievable with a GLS. This is because gate arrays are built from fully formed gates. These gates include additional large output drive transistors for supplying a specified fanout capability. In the GLS, these drive transistors are needed only at those outputs that must drive a large capacitance. The main savings therefore is in the area occupied by the drive transistors.

Choosing which architecture to use for a given function when a choice exists may be answered in part by timing considerations. For example, consider the two architectures in Figs. 9a and 9b for a four-bit parity encoder. The circuit in Fig. 9a is a two-level network, and Fig. 9b is a tree. Intuitively, it might seem reasonable to expect the circuit in Fig. 9a to have less delay from input to output than the circuit in Fig. 9b. This seems reasonable because there are fewer gates in the signal path of Fig. 9a than Fig. 9b. Unfortunately, this intuitive estimation is not always correct because the gates in Fig. 9a are larger and therefore slower than those in Fig. 9b.

Specifically, consider an NMOS GLS implementation of the circuits in Figs. 9a and 9b. A very crude approximation will be used to calculate the expected delays through these circuits. Assume it is possible to lump all capacitance seen by the drain of a transistor into a single term, which is designated  $c_p$ . This capacitance includes all parasitics due to increased metal length to accommodate the transistor, diffusion capacitance, Miller capacitance, etc. Additionally, assume that each pull-down transistor has a gate capacitance  $c_g$ . Define  $k$  as the ratio of pull-up resistor impedance to pull-down transistor impedance, and  $\tau$  as the transit time.

The capacitance and delay for each level in Figs. 9a and 9b are tabulated below, assuming an initial down transition at node 1.

9a		
Node	Capacitance	Delay
1	$5c_g + c_p$	$\tau(5c_g + c_p)/c_g$
2	$4c_g + c_p$	$k\tau(4c_g + c_p)/c_g$
3	$c_g + 4c_p$	$\tau(c_g + 4c_p)/c_g$
4	$c_g + 8c_p$	$k\tau(c_g + 8c_p)/c_g$

Total delay assuming an initial down transition at node 1 is:

$$\tau(6c_g + 5c_p)/c_g + k\tau(5c_g + 9c_p)/c_g$$

Similarly, the delay assuming an initial up transition at node 1 is:

$$k\tau(6c_g + 5c_p)/c_g + \tau(5c_g + 9c_p)/c_g$$

9b		
Node	Capacitance	Delay
1	$2c_g + c_p$	$\tau(2c_g + c_p)/c_g$
2	$c_g + c_p$	$k\tau(c_g + c_p)/c_g$
3	$c_g + 2c_p$	$\tau(c_g + 2c_p)/c_g$
4	$2c_g + 2c_p$	$2k\tau(c_g + c_p)/c_g$
5	$c_g + c_p$	$\tau(c_g + c_p)/c_g$
6	$c_g + 2c_p$	$k\tau(c_g + 2c_p)/c_g$
7	$c_g + 2c_p$	$\tau(c_g + 2c_p)/c_g$

Total delay assuming an initial down transition at node 1 is:

$$\tau(5c_g + 6c_p)/c_g + k\tau(4c_g + 5c_p)/c_g$$

The corresponding delay for an initial up transition at node 1 is:

$$k\tau(5c_g + 6c_p)/c_g + \tau(4c_g + 5c_p)/c_g$$

Assume  $c_p$  is on the order of  $c_g$  and that a typical value for  $k$  is 4. Then an examination of both sets of delays for both circuits reveals that Fig. 9a is always slower than Fig. 9b.

This is a very crude analysis intended only to show that the optimum choice of organizations based on delay is not always obvious. A rigorous analysis using actual layout parameters is the only way to accurately predict actual circuit delays. Performing such an analysis for several potential structures is only practical if an automated design system is available.

Another important factor to be considered when selecting the best architecture is topology. For example, the structure resulting from Fig. 9a will tend to have longer columns than Fig. 9b to accommodate the larger gates. However, Fig. 9b will tend to have longer implicant rows than Fig. 9a to accommodate more terms. A designer must decide which architecture best fits into the available area. As is the case for analyzing the timing of candidate architectures, determining dimensions of several structures is greatly facilitated by an automated design system.

## VI. Comments Regarding an Automated Design System

It is possible to design an automated system for mapping functions into a GLS. This system is probably best implemented on an interactive color graphics system. In this system, a user could specify the function to be mapped, design guidelines, and analysis to be performed. By viewing the GLS created by the computer, or the results of an analysis, the user

could suggest other mapping approaches or request changes be made.

The program might work by initially creating a copy of a clean GLS in memory. Cuts and contact points would be overlaid on this copy. An algorithm similar to printed circuit layout algorithms would be invoked to do the mapping iterations. The program would stop either when a desired set of requirements has been met or when no solution has been found after a time limit has been exceeded.

## References

1. Patil, S. S., and Williams, T., "An Approach to Using VLSI in Digital Systems," 5th Symp. Comp. Arch., 1978, pp. 139-143.
2. Eichelberger, E. B., and Williams, T. W., "A Logic Design Structure for LSI Testability," Proc. 14th Design Automation Conf., June 1977, pp. 462-467.

## Bibliography

Mead, C., and Conway, L., *Introduction to VLSI Systems* (text in preparation), 1978.

Sievers, M. W., "A General Logic Structure for Custom LSI," Jet Propulsion Laboratory IOM 331-78-109a, Oct. 5, 1978 (an internal document).

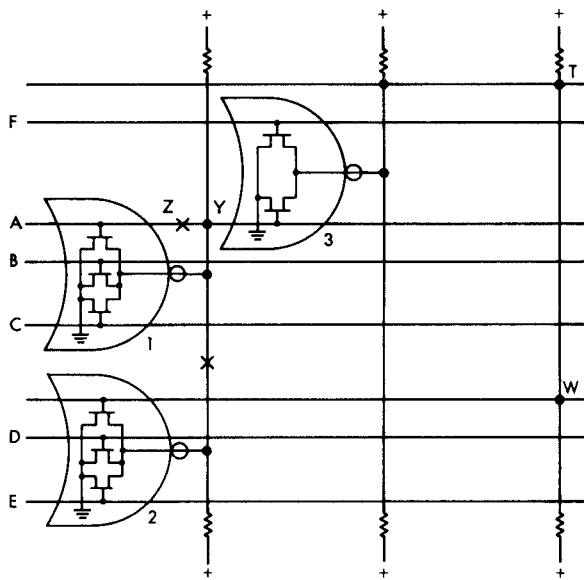


Fig. 1. Approximate logical equivalent of a GLS

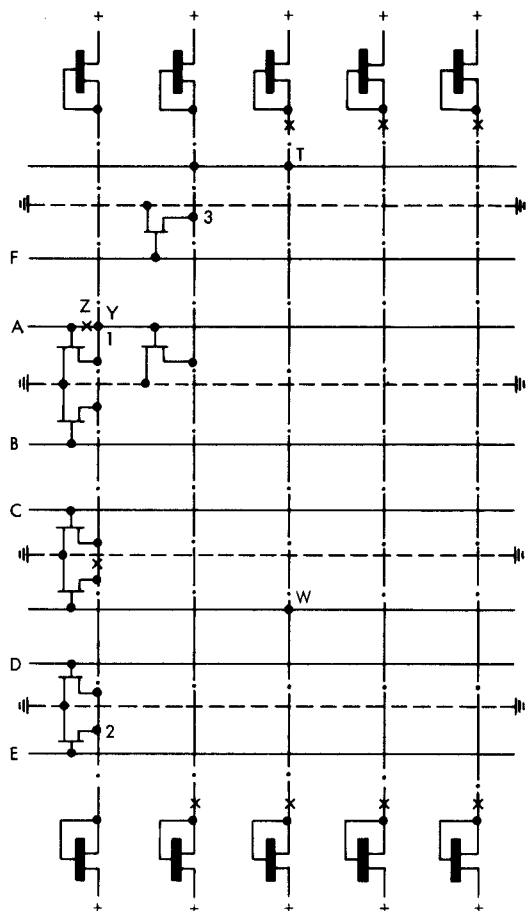


Fig. 2. Transistor level description of GLS example

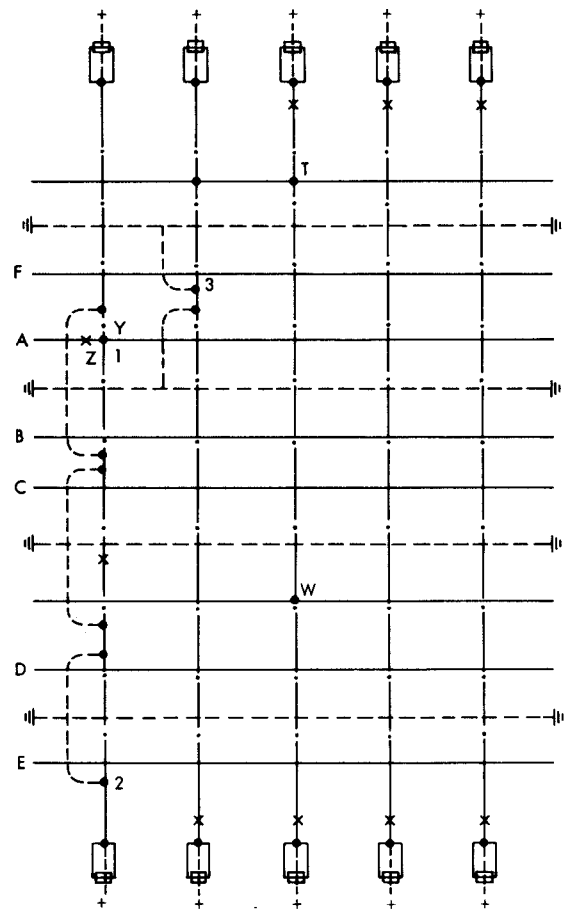


Fig. 3. NMOS stick diagram of GLS example

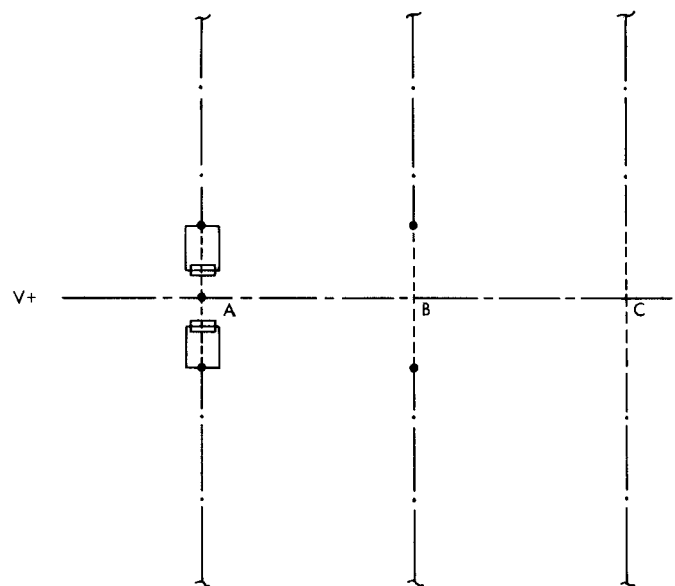


Fig. 4. Method for increasing number of pull-up resistors in column

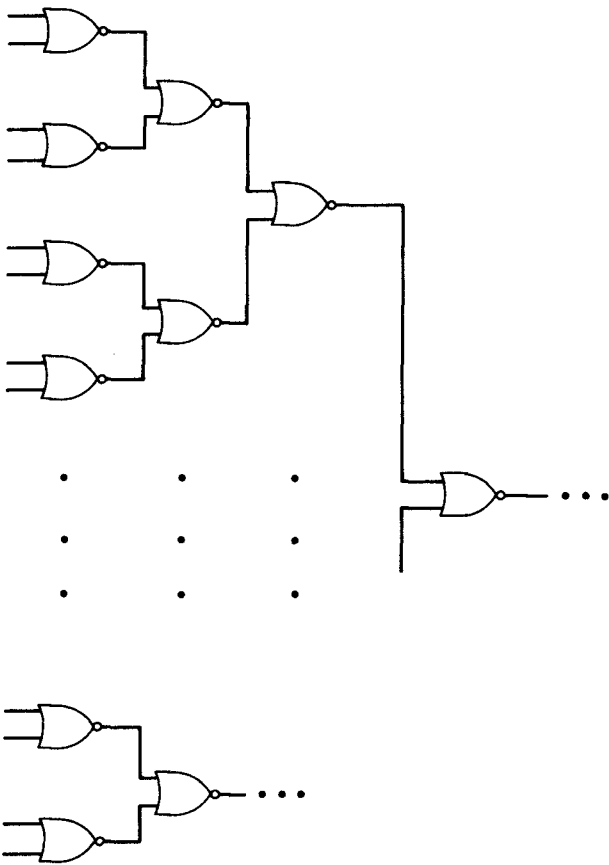


Fig. 5. Binary NOR tree

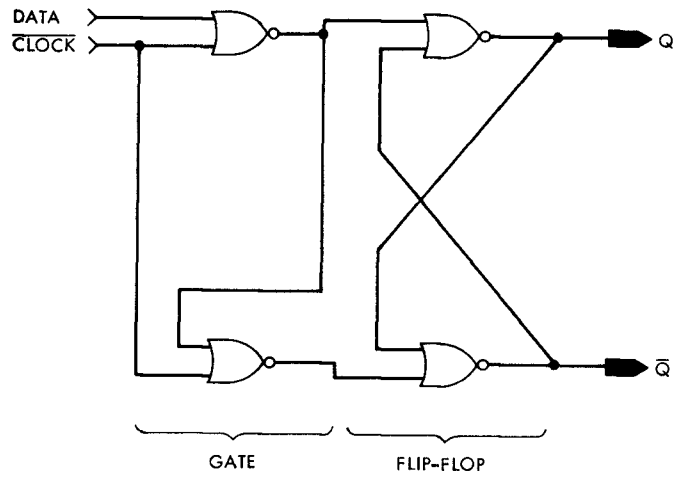


Fig. 6a. Logical equivalent of D flip-flop

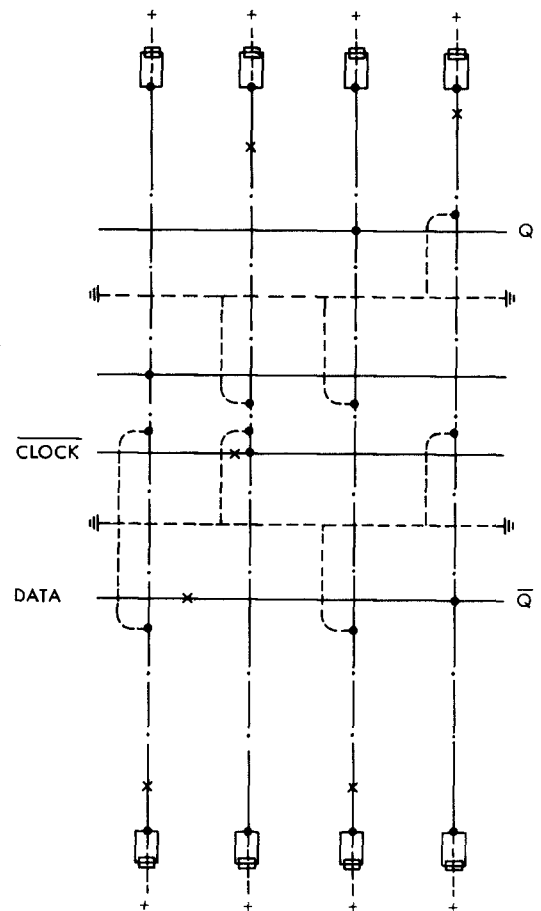


Fig. 6b. NMOS stick diagram of D flip-flop

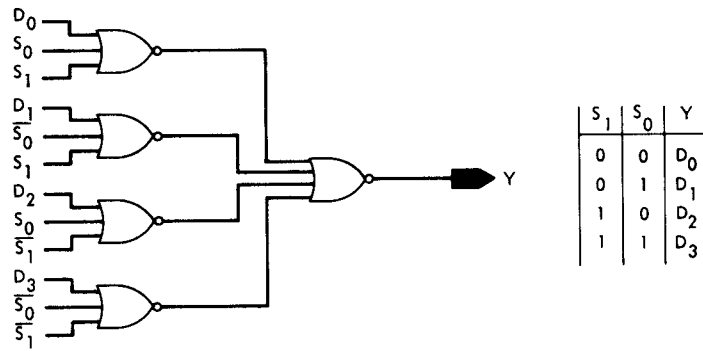


Fig. 7. Multiplexer implemented in NOR gates

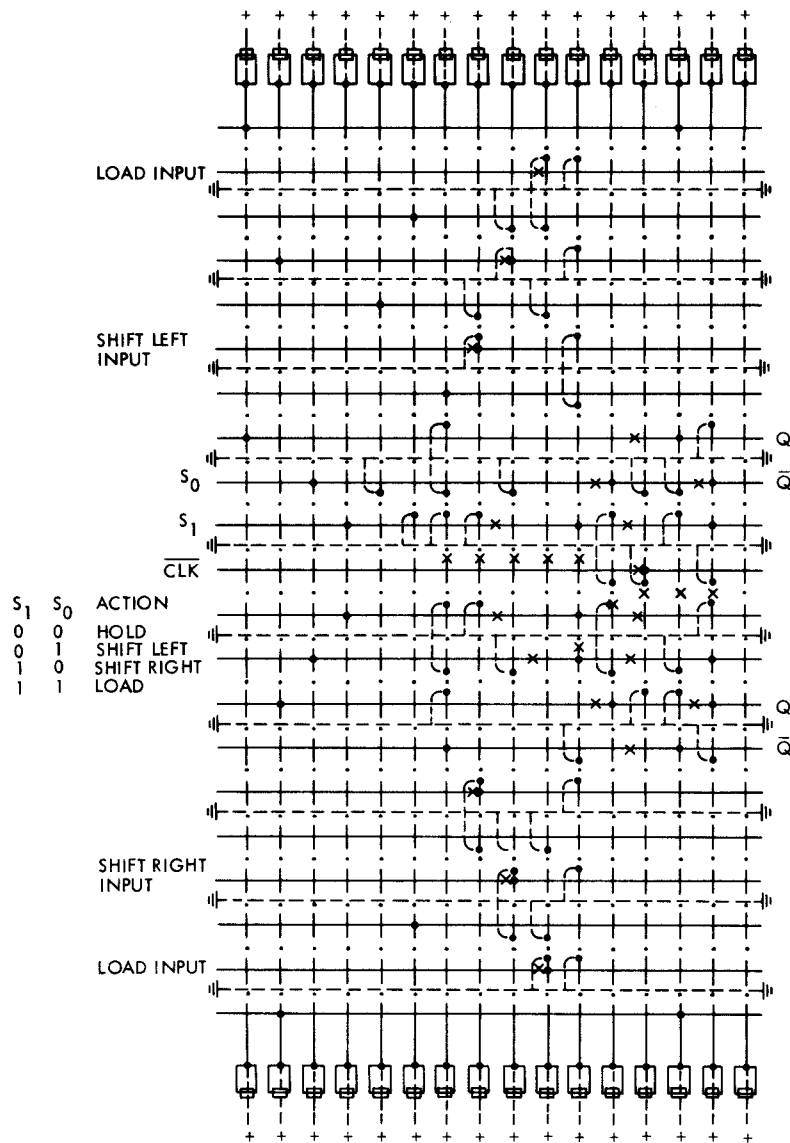


Fig. 8. Two cell shift register

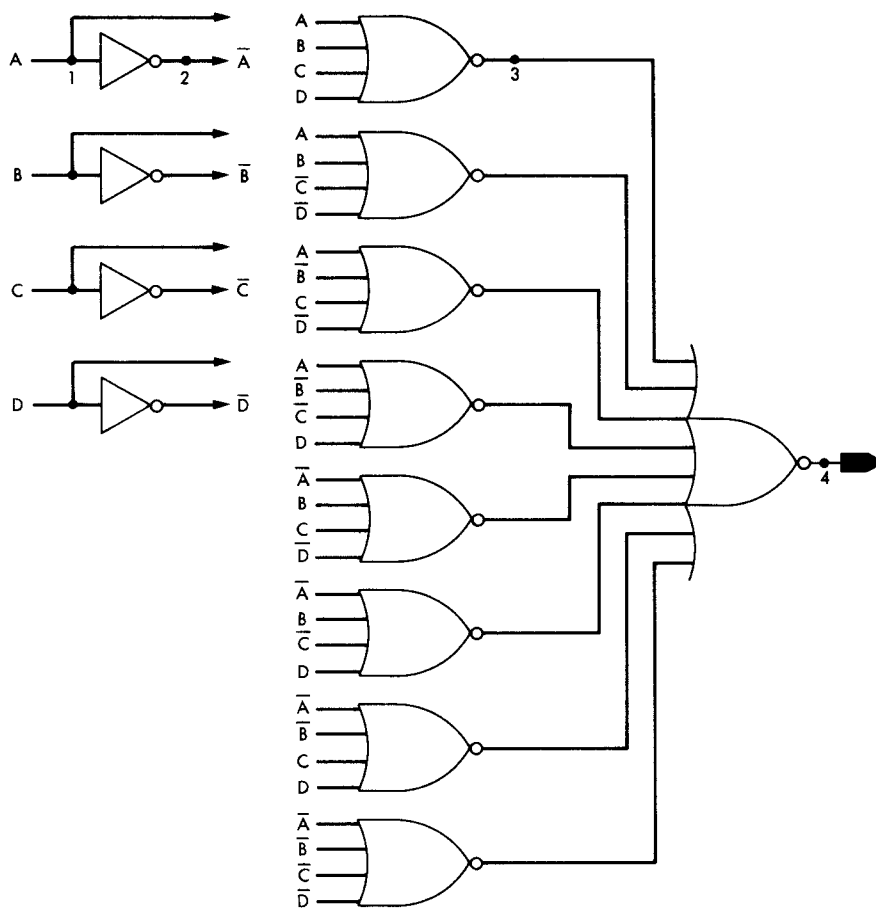


Fig. 9a. Two-level parity generator

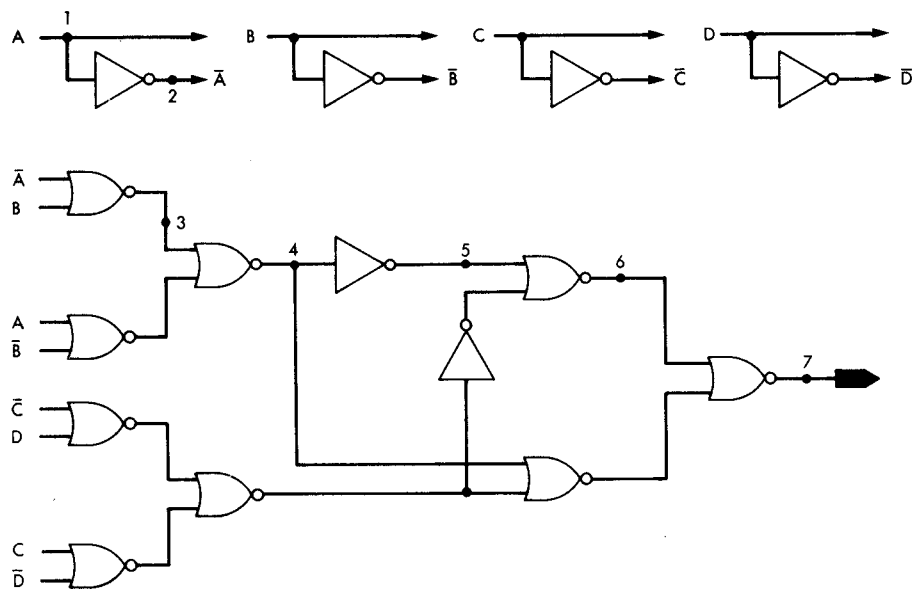


Fig. 9b. Parity generator tree